KEKER & VAN NEST LLP
ROBERT A. VAN NEST - #84065
rvannest@kvn.com
CHRISTA M. ANDERSON - #184325
canderson@kvn.com
633 Battery Street
San Francisco, CA 94111-1809
Telephone: 415.391.5400
Facsimile: 415.397.7188

KING & SPALDING LLP
DONALD F. ZIMMER, JR. - #112279
fzimmer@kslaw.com
CHERYL A. SABNIS - #224323
csabnis@kslaw.com
101 Second St., Suite 2300
San Francisco, CA 94105
Telephone: 415.318.1200
Facsimile: 415.318.1300

KING & SPALDING LLP
SCOTT T. WEINGAERTNER (*Pro Hac Vice)*
sweingaertner@kslaw.com
ROBERT F. PERRY
rperry@kslaw.com
BRUCE W. BABER (*Pro Hac Vice)*
1185 Avenue of the Americas
New York, NY 10036
Telephone: 212.556.2100
Facsimile: 212.556.2222

GREENBERG TRAURIG, LLP
IAN C. BALLON - #141819
ballon@gtlaw.com
HEATHER MEEKER - #172148
meekerh@gtlaw.com
1900 University Avenue
East Palo Alto, CA 94303
Telephone: 650.328.8500
Facsimile: 650.328.8508

Attorneys for Defendant
GOOGLE INC.

UNITED STATES DISTRICT COURT

NORTHERN DISTRICT OF CALIFORNIA

SAN FRANCISCO DIVISION

ORACLE AMERICA, INC.,

Plaintiff,

v.

GOOGLE INC.,

Defendant.

Case No. 3:10-cv-03561-WHA

REPLY DECLARATION OF OWEN ASTRACHAN IN SUPPORT OF DEFENDANT GOOGLE INC.'S MOTION FOR SUMMARY JUDGMENT ON COUNT VIII OF PLAINTIFF ORACLE AMERICA'S AMENDED COMPLAINT

Judge: Hon. William Alsup

Hearing: 2:00 p.m., September 15, 2011

I, Owen Astrachan, declare as follows:

1. I am the same Owen Astrachan who submitted an August 1, 2011 Declaration in support of Defendant Google Inc.'s Motion for Summary Judgment on Count VIII of Plaintiff Oracle America, Inc.'s Amended Complaint. I submit this reply declaration in support of Defendant Google Inc.'s Motion for Summary Judgment on Count VIII of Plaintiff Oracle America, Inc.'s Amended Complaint. I make this declaration based on my own personal knowledge. If called as a witness, I could and would testify competently to the matters set forth herein.

2. Attached hereto as Exhibit 3 (for ease of reference, I am not reusing exhibit numbers that were used in my prior declaration) is a true and correct copy of the rebuttal expert report ("Astrachan Rebuttal Report") I prepared in this action.

3. Attached hereto as Exhibit 4 is a true and correct copy of the rebuttal expert report ("Astrachan Reply Report") I prepared in this action.

4. The Astrachan Rebuttal Report and the Astrachan Reply Report are true and correct expressions of my opinions based on the facts I currently know.

I declare under penalty of perjury that the foregoing facts are true and correct. Executed on August 29, 2011 in Durham, North Carolina.

Owen Astrachan

EXHIBIT 3

SEPARATELY LODGED UNDER SEAL

EXHIBIT 4

ROBERT A. VAN NEST — #84065
rvannest@kvn.com
CHRISTA M. ANDERSON — #184325
canderson@kvn.com
KEKER & VAN NEST LLP
710 Sansome Street
San Francisco, CA 94111-1704
Telephone: (415) 391-5400
Facsimile: (415) 397-7188

DONALD F. ZIMMER, JR. (SBN 112279)
fzimmer@kslaw.com
CHERYL A. SABNIS (SBN 224323)
csabnis@kslaw.com
KING & SPALDING LLP
101 Second Street – Suite 2300
San Francisco, CA 94105
Telephone: (415) 318-1200
Facsimile: (415) 318-1300

SCOTT T. WEINGAERTNER (*Pro Hac Vice*)
sweingaertner@kslaw.com
ROBERT F. PERRY
rperry@kslaw.com
BRUCE W. BABER (*Pro Hac Vice*)
bbaber@kslaw.com
KING & SPALDING LLP
1185 Avenue of the Americas
New York, NY 10036-4003
Telephone: (212) 556-2100
Facsimile: (212) 556-2222

IAN C. BALLON (SBN 141819)
ballon@gtlaw.com
VALERIE HO (SBN 200505)
hov@gtlaw.com
HEATHER MEEKER (SBN 172148)
meekerh@gtlaw.com
GREENBERG TRAURIG, LLP
1900 University Avenue
East Palo Alto, CA 94303
Telephone: (650) 328-8500
Facsimile: (650) 328-8508

Attorneys for Defendant
GOOGLE INC.

UNITED STATES DISTRICT COURT

NORTHERN DISTRICT OF CALIFORNIA

SAN FRANCISCO DIVISION

ORACLE AMERICA, INC.

Plaintiff,

v.

GOOGLE INC.

Defendant.

Case No. 3:10-cv-03561-WHA

Honorable Judge William Alsup

REPLY TO DR. JOHN C. MITCHELL'S OPPOSITION TO DR. OWEN ASTRACHAN'S OPENING EXPERT REPORT

I. INTRODUCTION

1. I have been asked by Google to review Dr. John C. Mitchell's Report In Opposition to Dr. Owen Astrachan's Opening Expert Report submitted on behalf of plaintiff Oracle America, Inc. in this action ("Opposition Report"), and to reply to the opinions and conclusions set forth in that report.

2. My qualifications and the opinions set forth in my July 29, 2011 Opening Expert Report ("Opening Report") and my August 12, 2011 Rebuttal Expert Report ("Rebuttal Report") are incorporated herein by reference.

3. I understand that I may be asked by Google to review further submissions related to copyright issues from Oracle's experts, and to provide my opinions on issues raised by any such submissions.

4. I understand that I may be called upon to testify in this case regarding my opinions and analyses set forth in this report. If called upon to testify, I may use various demonstratives, including tables or drawings, to assist in presenting my testimony.

5. As set forth in my Opening Report, my compensation does not depend in any way on the outcome of this litigation.

II. DOCUMENTS AND INFORMATION CONSIDERED

6. My opinions are based on my relevant knowledge and experience, the documents identified in Exhibit B to my Opening Report, the documents identified in Section II of my Rebuttal Report, as well as review of the following documents and information:

A. Dr. John C. Mitchell's Report In Opposition to Dr. Owen Astrachan's Opening Expert Report ("Opposition Report"), dated August 12, 2011.

B. "The Mythical Man-Month," Frederick Brooks, Anniversary Edition (1995).

III. BRIEF SUMMARY OF MY OPINIONS

7. Based upon my review of the material set forth in Section II, it is my opinion that Prof. Mitchell:

A. does not meaningfully dispute that there is no literal source code copying at issue, outside of 12 files; and

B. frequently confuses ideas with creative expression, and often uses incorrect or irrelevant examples when discussing creative expression; and

C. incorrectly characterizes or otherwise fails to rebut several of the arguments set forth in my Opening Report.

IV. THERE IS NO LITERAL SOURCE CODE COPYING AT ISSUE, OUTSIDE OF 12 FILES OUT OF THOUSANDS

8. Prof. Mitchell agrees that aside from portions of 12 files that I identified in paragraphs 150-177 of my Opening Report as qualitatively and quantitatively insignificant, Oracle does not allege that Google has engaged in literal copying of source code. Prof. Mitchell does not dispute that the allegedly copied portions of the 12 files at issue constitute less than 0.05% of the lines of code in Oracle Java 1.5, which is quantitatively *de minimis*. Although he disputes my assessment of the qualitative importance of these files, he makes no claim that these files are central to the functionality of either Java or Android.

9. The other alleged copying that Prof. Mitchell addresses is the alleged copying of 37 APIs, which are methods of operation and necessary for interoperability and compatibility. Specifically, Prof. Mitchell addresses the following aspects of the APIs at issue: the selection and arrangement (which is functional and constrained by factors such as efficiency, standard programming techniques, and design standards), certain aspects of the documentation (which are factual, descriptive, and constrained by requirements of brevity), and the method signatures and data fields (which are functional names, methods of operation, and constrained by factors such as efficiency, standard programming techniques, industry demand, and design standards).

10. Prof. Mitchell likewise does not dispute that the APIs at issue are methods of operation that programmers use to access, control and manipulate the underlying software functionality that is part of the Java and Android platforms.

V. PROFESSOR MITCHELL REPEATEDLY CONFUSES MAKING CHOICES AND CREATIVE EXPRESSION

11. Throughout his report, Prof. Mitchell points to various *choices* that were made during the design of the APIs at issue. (*See, e.g.*, Opposition Report ¶¶ 6, 24, 29, 30, 31, 35, 37, 38, 39, 40,

41, 43, 44, 53, 80, and specific examples addressed in paragraphs 32-38 in this report.) But making choices is not synonymous with creative expression.

12. It is my understanding is that copyright protection never extends to "any idea, procedure, process, system, method of operation, concept, principle, or discovery." Ideas, of course, require choices. So, too, do procedures, processes, systems, methods of operation, principles, and discoveries. But because copyright protection does not extend to these categories, *choices* that affect, for example, what procedures or methods of operation are made available to programmers via APIs cannot result in copyrightable subject matter.

VI. COMPLEXITY IS NOT THE SAME AS CREATIVE EXPRESSION

13. Prof. Mitchell describes the complexity of the Java APIs (Opposition Report ¶¶ 18-38), and then implies, without directly explaining why, that this complexity necessarily results in a conclusion that the APIs are protected by copyright.

14. First, Prof. Mitchell does not explain what he means by "complex." Nor does he explain why a "complex" interface is any less a method of operation than a simple interface.

15. Second, Prof. Mitchell confuses the "complexity" that he claims went into the designing of the APIs at issue with the "complexity" of the APIs themselves. Specifically, he states that my analysis largely views the APIs at issue "from the perspective of an application programmer using [the] completed API[s] rather than considering the art of creating new APIs." (Opposition Report ¶ 20.) But the alleged copyright infringement is based on the APIs themselves, not the process of creating those APIs.

16. As Prof. Mitchell acknowledges, APIs are an *abstraction*. (Opposition Report ¶ 23.) That is, they are *less* complex than the code that implements the APIs. In fact, the APIs themselves are not executable code. That is, APIs are not, themselves, software. They are shortcuts, or labels, that refer to the underlying software, sort of like a system of car parts in a catalog might have names or numbers that are then used by a mechanic to actually find and then use the underlying parts. When software is compiled, these labels are used to access the actual software that implements the functionality, but the labels (the APIs) do not implement the functionality themselves. Indeed, Prof. Mitchell states that the source code that implements an

API is "a written expression of the API." (Opposition Report ¶ 55.) The APIs represent *ideas*, while any creative expression is in the *implementing code*.

VII. PROFESSOR MITCHELL'S ARGUMENTS IN SUPPORT OF HIS CONCLUSION OF CREATIVE EXPRESSION LARGELY FOCUS ON *FUNCTIONAL* CONSIDERATIONS

17. Although Prof. Mitchell claims that the structure and organization of the APIs was not driven by function, his explanation of what drove the choices is at odds with his own conclusion. For example, he says that the choice of which APIs to include was a choice of which "tools" (*i.e.* functions) to make available to developers. (Opposition Report ¶ 20.) He also states that parameters were chosen to enhance stability. (Opposition Report ¶ 22.)

18. Other aspects of the APIs organization are also functional, according to Prof. Mitchell's explanations. For example, he says that the APIs were organized as a set of consistent, logical, and easy to follow rules. (Opposition Report ¶ 23.) The organization chosen for methods has "important practical [*i.e.* functional] implications." (Opposition Report ¶ 25.) Features that are shared across many classes should be defined as Interfaces rather than classes. (Opposition Report ¶ 27.) My own Opening and Rebuttal Reports also explain why the organization is constrained by factors such as efficiency, standard programming techniques, and design standards. For example, as I pointed out in paragraph 45 of my Rebuttal Report, Prof. Mitchell's original example of the flexibility of structure and organization was in fact almost certainly a textbook use of a standard and widely used design pattern. Other pragmatic requirements (including familiarity with the organization of other languages and requirements of extensibility) may also substantially constrain any creativity involved in the expression of this organization.

19. As a practical matter, developers typically write code and software interfaces so that they are efficient, and comport with industry demands and widely accepted practices. They also typically choose organizational strategies used in systems that are in the public domain. The selection, arrangement and organization of the APIs at issue reflect these practices. They provide an organizational framework that is functional, not creatively expressive.

20. These functional considerations reinforce my opinion that the APIs are not creative

expression.

VIII. PROFESSOR MITCHELL LARGELY IGNORES THE ISSUE OF COMPATIBILITY AND INTEROPERABILITY

21. Prof. Mitchell says that "the Java APIs differ fundamentally from, for example, an interface that is required in order for a video game to play on a particular game machine platform. In the latter case, if the proper interface is not in place, the video game will not be able to play on the platform. But a software developer can program in Java without ever using any particular library classes that are implementable in the Java language." (Opposition Report ¶ 21.)

22. First, this ignores Oracle's own documentation, which for example describes the classes in the java.lang package (one of the API packages at issue) as "fundamental to the design of the Java programming language." Thus, asking a software developer to program in the Java programming language without using any of the APIs at issue is asking that programmer to ignore aspects of the language that Oracle itself has called "fundamental." This is like asking someone to write in the English language but then denying them the ability to use common, well-known English words.

23. Second, even if, theoretically speaking, developers could program in Java without using the APIs at issue, implementing standard and well known class libraries serves the important benefit of increasing compatibility. In this respect, the Java APIs are precisely the same as the interfaces for a particular game machine. As Prof. Mitchell explains, in the case of a game machine, "if the proper interface is not in place, the video game will not be able to play on the platform." Similarly, in the case of the APIs at issue, if the APIs are not in place, code written to use those APIs will not work.

24. Prof. Mitchell also states that he disagrees with my conclusion that use of the APIs at issue was necessary to ensure basic functionality and interoperability. (Opposition Report ¶ 56.) According to Prof. Mitchell, Google could instead have implemented different APIs. (Opposition Report ¶ 57.) Had Google done so, however, code written for the Android APIs would not interoperate with code written for Oracle's APIs.

25. Prof. Mitchell also claims that because Google did not implement *all* of Oracle's APIs, that somehow means that the goal of interoperability was not served by using the APIs that Google did implement. (Opposition Report ¶¶ 61-70.) I disagree. The goal of interoperability is furthered by implementing already known APIs. If Google implemented more of the APIs, code for the Android and Java platforms would be even more interoperable. But Google presumably had multiple design goals, of which interoperability was only one. One such goal might well have been implementing a user interface that was highly intuitive for mobile smartphone users, which would explain why Android includes user interface APIs that differ from the user interface APIs in Oracle's desktop or pre-smartphone oriented Java platform. Regardless, by implementing the Java language APIs that it did, Google enhanced Android's compatiblility with code written for the Java platform and vice versa, thereby enhancing the interoperability between the two platforms.

26. Importantly, the goal of interoperability requires not just that the method declarations be the same, but that they be organized the same way. If the methods had been organized in different ways, this would have had, to use Prof. Mitchell's words, "important practical implications" (Opposition Report ¶ 25), and would have prevented interoperability.

IX. PROFESSOR MITCHELL'S DISCUSSION OF "SIMILAR" DOCUMENTATION AND COMMENTS IGNORES THE OBVIOUS REASON FOR THAT SIMILARITY

27. Prof. Mitchell states that he disagrees with my opinion that the Android API documentation is not substantially similar to Oracle's API documentation. (Opposition Report ¶ 52.) He notes that the "left hand column" of the Android documentation is "virtually identical" to the Oracle documentation. Prof. Mitchell ignores the fact that the left hand column contains the items that are being documented, and both sets of documentation document all the APIs at issue. Prof. Mitchell's observation is like noting that two dictionaries share a very high number of words in bold face at the left of each column (*i.e.* the words being defined), and concluding that copyright infringement has occurred.

28. Prof. Mitchell also states that "a very high percentage of the English language text

descriptions contained in the Android appear to have been written to closely track the text descriptions in the Java APIs." (Opposition Report ¶ 54.) He does not offer any examples of this purported close tracking, but my own review of the two sets of documentation leads me to precisely the opposite conclusion.

29. Prof. Mitchell also states that comments in the Android source code generally are "quite similar" to the prose descriptions in the Java API specifications. (Opposition Report ¶ 75.) He offers no particular examples in support of this assertion. As noted in paragraphs 145 to 149 of my Opening Report, any apparent similarities in the documentation that I examined appear to be explained by the functional requirement that the documentation be factual and descriptive. Moreover, he ignores the fact that the texts that he is comparing are *describing the same APIs*. Not surprisingly, the descriptions share some similarities.

X. PROF. MITCHELL'S EXAMPLES OF THE CREATIVITY OF APIs ARE MISLEADING

30. Prof. Mitchell's Opposition Report uses several examples that purport to show that the creation of the Java APIs involved creativity, but many of these examples are misleading. For example, he discusses two different ways to create a warning dialog in Java and Android, by stating that two different APIs for creating a button exist. (Opposition Report ¶¶ 63-69.) Again, it is my understanding that selecting between two design alternatives is not the correct standard to use to evaluate whether something is creative expression. In the case of the example of the warning dialog, it is likely that the Java APIs used for windows and dialogs were replaced in order to reflect the very different screens and text entry models used by modern mobile devices.

31. Similarly, Prof. Mitchell notes that alternative libraries exist for some of the functionalities covered by the core API, usually predating Java's implementation, and implying that Android could feasibly have used those alternative APIs instead. (Opposition Report ¶ 59.) Again, while providing an example demonstrating that selection was possible, Prof. Mitchell's analysis does not address the reasons why two different versions might exist, or why such a selection might be made. In this case, many of these alternative APIs cease to be developed when the functionality is finally included in the Java platform. Once that happens, industry

demand for compatibility and skill reuse pushes individual and corporate developers to use the official version, and as discussed in my Opening Report, it is my understanding that these factors can weigh against the protectability of a work. Prof. Mitchell's focus on selection, rather than a more detailed analysis of why such selections were made, does not provide the proper information necessary for an assessment of the creativity of a given API.

32. Prof. Mitchell quotes Bob Lee as saying that he would "much rather be designing APIs than re-implementing them." (Opposition Report ¶ 5.) He uses this to imply that Mr. Lee's preferred activity (API design) must result in creative expression. The fact that some engineers enjoy developing APIs more than writing the code that implements them does not mean that APIs are more creatively expressive than the underlying code. Instead, in my opinion, this merely reflects that some engineers prefer coming up with ideas (or systems, or methods of operation), while some prefer the different challenges that come with coding, such as optimizing for performance.

33. In another example, paragraph 22 of Prof. Mitchell's Opposition Report seems to suggest that the number and type of parameters available to the designer of an API are extremely broad and flexible, so that any resulting organization and selection is extremely creative. While programmers certainly do make choices about the number and nature of the parameters of a given API, the resulting expression is still extremely constrained, because it must directly reflect the idea of the underlying functionality of the program. To put it another way, in my experience, programmers first conceive the idea of what a function will do, and then they decide on the appropriate parameters to achieve that functionality. The resulting set of parameters will reflect and be constrained by the requirements of efficiency and this underlying functionality. For example, in paragraph 37, Prof. Mitchell gives the example of the "ulp" method in Java and the "float_distance" function in Boost. The ulp method in Java takes one parameter, and then compares that parameter to a another number determined by the parameter. This number is calculated by the ulp method. The float_distance function instead takes two parameters, and compares them to each other. It is true that one of these methods has one parameter, and the other has two, but this is because the two methods have different functionality. Both compute

different values, and therefore must have a different number of parameters. These methods are both related to so-called *unit of least precision* or *unit in last place*, but the methods compute different, though related values, which is why a different number of parameters are used. The Math.ulp function returns the distance (measured in units called ulps) to the next representable floating point number after the parameter. The Boost function instead returns the distance between its two parameters. As a result, this example is simply incorrect — the different parameters reflect different underlying functionality rather than creativity.

34. Prof. Mitchell says that the Comparable interface is a "clever combination of lessthan, greaterthan, equal to." (Opposition Report ¶ 27.) This is incorrect. Comparable (and specifically compareTo) is essentially a standard "strcmp" function. Strcmp is short for "string comparison," and like compareTo it compares two things and returns an integer indicating less than, greater than, or equal to. Strcmp was present in standard C and C++ libraries well before the existence of Java, and documented at least as early as Kernighan and Ritchie's "The C Programming Language" in 1978. When the function was brought into Java, it was adjusted to make it object oriented, but the basic semantics (returning zero if the objects were equivalent, a number less than zero if the first object was less than the second object number, and a number more than zero if the first object was more than the second object) were preserved. Comparable's use of Interface, like Comparable's semantics, is not "elegant and creative," but instead is necessary to allow other parts of Java to make use of this functionality in a standardized way known as "inheritance." Inheritance dates back to the earliest object-oriented language (Simula), in the 1960s, and the Java language uses Interface as part of its implementation of inheritance. So use of Interface is not creative; instead, Interface is a well-understood technique to solve the well-understood problem of providing the same functionality in multiple places. As a result, I understand that this should not be copyrightable subject matter.

35. Prof. Mitchell's assertion that designing a class hierarchy is "nontrivial" (Opposition Report ¶ 32) is also misleading. Classes extend across package boundaries not because of a creative impulse, but to capture functionality that is spread across multiple packages. Prof. Mitchell implies that designers could have chosen to resolve this problem in another way, by

saying that "[h]ad designers been following some rule…" they might have chosen a different mechanism. However, no such rule exists, because such a rule would be inefficient and nonsensical., Instead, basic functionality and efficiency create basic rules that must generally be sensibly followed, and so the resulting expression does not represent creativity.

36. Prof. Mitchell tries to show that the designers of the java.util API made a "conscious design choice" when they placed two "exceptions" called java.util.EmptyStackException and java.util.NoSuchElementException in the java.util package instead of the java.lang package. (Opposition Report ¶ 35.) He bases this argument on the fact that these exceptions "inherit from" (*i.e.*, are based on) a member of the java.lang package called java.lang.Exception. In fact, as part of standard software design principles, exceptions and classes are essentially always defined in the package they logically belong to, so that programmers can find them and use them more efficiently. As a result, this subclassing of Exception in places other than java.lang is the rule, not the exception: there are roughly 16 subclasses of Exception defined in the java.io package, another 16 in java.security, 12 in java.net, and more in other packages. This strongly suggests that the API designers were using standard design practices and responding to the functional needs of users of the API when creating Exception subclasses as part of the package they originate in, rather than any sophisticated and considered "conscious design choice" as Prof. Mitchell implies.

37. Prof. Mitchell also argues that making java.util.Enumeration and java.util.Observer Interfaces rather than Classes represents a significant design choice on the part of the authors of the java.util API. Again, these choices were dictated primarily by standard software design principles, and particularly by a desire for efficient, flexible interfaces. Interfaces provide greater flexibility to developers, because a class can implement several Interfaces, but extend only one class. As a result, there is a standard design principle that unless there is code shared by all classes that will implement an Interface, an Interface should be used instead of an abstract class. Where code is shared, an abstract class that implements the Interface should be used to make the programming task simpler and easier. In the example of Enumeration, there is no code that can be shared among all the classes that implement the Enumeration interface (or the more

modern Java equivalent, Iterator.) The same is true of the Observer interface. As a result, it is very standard design practice to make them an Interface instead of a Class. In contrast, the Observable class is always used with objects that implement the Observer interface, and as a result, it is implemented as an abstract class rather than an Interface because there is code to be shared among all Observable implementations. Java's consistent compliance with this basic design principle can be seen in other examples as well, such as in java.util, where the List interface is implemented by the class AbstractList and the Collection interface is implemented by the class AbstractCollection. In each of these places, the designers of the API were following standard design principles used industry-wide. This is design pragmatism rather than creativity.

38. Prof. Mitchell suggests that certain private fields, such as codeSource, were copied, even though they are not part of the public API. (Opposition Report ¶ 50.) This is misleading. It is true that a private field called "codeSource" exists in the file at issue, but, as Prof. Mitchell notes, for each of the private fields he discusses, there is also a public method called "getCodeSource." The Java language's convention is that, when a public method is called "getX," there is then a private field called "X." To say that the existence of the "X" private field proves that X was copied is akin to saying that my use of Prof. Mitchell in this report proves that I copied from another document that referred to him as Prof. Mitchell, when in fact it is much more likely that it proves that I knew the social convention of using "Prof." as an honorific. Similarly, the use of the private field "codeSource" here does not imply copying or creativity, as Prof. Mitchell seems to suggest. Instead, it implies that whoever wrote the Android implementation of this package followed pragmatic conventions to create private field names from public method names.

XI. PROF. MITCHELL MISCHARACTERIZES MY ARGUMENTS, AND FAILS TO REBUT MY CORE CONCLUSIONS

A. PROF. MITCHELL'S CRITIQUES OF MY API ANALOGIES OBSCURE, BUT DO NOT REBUT, THE POINT OF THOSE ANALOGIES

39. Prof. Mitchell attacks my Opening Report's analogies between APIs and cars, and between APIs and menu shortcuts such as "Ctl+C" and "Ctl+P." (Opposition Report ¶ 18.) Specifically, Prof. Mitchell states that because the Java APIs are more complex than a car or

software menu system, the analogies do not meaningfully describe the APIs at issue. Even if the APIs at issue are more complex than a car or software menu shortcut, these analogies are still relevant.

40. First, Mitchell's focus on the comparable complexity of a car or menu shortcut to an API does not rebut my conclusion that APIs are methods of operating underlying functionality. The API that allows a programmer to control a complex piece of software may itself be more or less complex, but this does not change the purpose of the API: allowing communication between two pieces of software so that one can control the other. It is perhaps true that a car's "API" may be less complex than some software APIs, in which case maybe a plane's cockpit may be a more accurate analogy than a car. However, whether discussing the car or the plane, the levers, steering mechanisms, and dials involved are still simplified mechanisms that allow humans (and the software they write) to operate complex, sophisticated underlying functionality, and Prof. Mitchell does not dispute this conclusion. Although Prof. Mitchell argues that in some cases developers will have to understand complex systems in order to operate them (Opposition Report ¶ 28), he does not argue that it should be necessary to look "under the hood." He merely states that, when the underlying system is complex, more effort must be taken to understand the methods for operating the system.

41. Prof. Mitchell does not dispute the other major point of these analogies, which is that APIs are necessary for interoperability between software. APIs allow for standardized mechanisms that can be reused by programmers and their software, just like users learn Ctl+P and use it consistently across many programs. This principle holds true, regardless of the level of complexity of the API. If anything, the level of complexity of the underlying system makes the APIs *more* important to interoperability and compatibility. Programmers, having learned a complex API, are less likely to learn other APIs, making it more important that new entrants to a given market provide compatible APIs.

42. To better understand this situation, it may help to be reminded again that APIs are not themselves functional software. They are shortcuts, or labels, that refer to the underlying software. When software is compiled, these labels are used to reference the actual software that

implements the functionality, but do not implement the functionality themselves. This is true whether the API contains 10 elements or 10,000, and Prof. Mitchell has not contradicted this fact.

43. For these reasons, Prof. Mitchell does not effectively rebut the most important point of these analogies, which is that APIs are methods of operating underlying functionality, and that APIs are necessary for interoperability and compatibility.

B. PROF. MITCHELL INCORRECTLY STATES THAT I THINK THE LENGTH OF NAMES INHERENTLY MAKES THEM UNCREATIVE

44. Prof. Mitchell says that it "does not make sense to parse [names] into average word length and claim that they are not original." (Opposition Report ¶ 42.) My goal in analyzing these statistics was primarily to show that the functional API elements at issue are words and short phrases, and specifically short names, not to make a judgment on their creativity directly. It is my understanding that words and short phrases, such as names, are not protectable. Prof. Mitchell has not contradicted the underlying analysis: most of the API elements at issue here are single words or short phrases of 1-2 words, and they name the underlying functionality. It is my understanding that as a result they are not protectable. In addition, Prof. Mitchell's own examples of purportedly creative names are not examples of creative expression. For example, in paragraph 40, he gives "Proxy" in java.net as an example of a name that is not closely tied to its associated function. The underlying functionality being labeled in java.net's Proxy class is a setting for a proxy, so the name Proxy is very clearly tied to the underlying functionality. Similarly, he suggests that two elements in the Iterator Interface called "next" and "hasNext" could have, instead, been called "hasNextElement" and "getNextElement" instead. It is true that either set of names would have been descriptive. However, both sets of names are constrained by the underlying functionality, as they must be in order for them to be useful to programmers. The names and design of the Iterator Interface were also inspired by the Iterator design pattern, which predates Java. Finally, Prof. Mitchell's own source says that these names were chosen because they "fit neatly on one line" (Opposition Report ¶ 40), *i.e.*, that the names were chosen because they were short and efficient, leading to code that is easier to write, and easier to read

and maintain in the long run. Prof. Mitchell glosses over this fact, referring to the names instead as "aesthetically pleasing." Nowhere does he offer an example of a long name, or a name whose meaning is radically divorced from the underlying functionality. It is likely that there are no such names because they would be bad programming practice.

C. PROF. MITCHELL INCORRECTLY STATES THAT I THINK APIs ARE ONLY FOR COMPUTERS

45. Prof. Mitchell says that I think APIs are only for computers, but that APIs are "geared towards humans." (Opposition Report ¶ 23.) It is of course true that APIs are geared towards humans. In fact, in my Opening Report, I analyze in paragraph 131 why APIs are so important to interoperability and compatibility, discussing at length why it is important that humans — *i.e.*, programmers — be able to reuse their knowledge and their code through the use of APIs. I also discussed extensively (for example, in paragraph 118 of my Opening Report) why it is important that, for the benefit of humans, names be brief, efficient, and directly tied to their functionality. These characteristics are necessary because of the humans who will read, study, and use the APIs. However, whether or not APIs are "geared towards humans" is irrelevant to the questions of creative expression and API protectability that is central to this case.

D. PROF. MITCHELL DRAWS UNSUBSTANTIATED OR CONFLICTING CONCLUSIONS

46. Prof. Mitchell draws many unsubstantiated or logically conflicting conclusions. For example, Prof. Mitchell repeatedly attributes the success of Android to Android's use of Java, but then also notes that the only similarly successful platform (iPhone) uses a completely different language, Objective C. Statistics used in his opening report also suggest that another mobile platform that incorporates Java, BlackBerry, is failing in the marketplace. The data provided by Prof. Mitchell on the success of iPhone (without Java) and failure of BlackBerry (with Java) seem to contradict his own argument that Java is the key driver for Android's success. This strongly suggests that the use of the 37 Java API packages at issue is unlikely to be the reason for Android's success, and that other factors (such as Android's simple user interface, excellent web browser, built-in music player, or Google's strong brand) might be as or more

important than the use of Java. However, except for a throwaway mention of "the features shared by Android and iPhone" (which, notably, do not include Java), Prof. Mitchell never analyzes or even mentions these other features. These simple logical errors should call into question his entire analysis of the sources of market demand for Android.

47. Prof. Mitchell also asserts, without apparent justification, that Android "deliberately" fragmented Java. (Opposition Report ¶ 103.) Similar to his assertions about the source of the Java documentation, Prof. Mitchell does not provide any information to justify this claim or to allow me to directly dispute it.

48. Another example of Prof. Mitchell's unsubstantiated conclusions is his analysis of rangeCheck. (Opposition Report ¶ 86.) This code is only nine lines long, and used in only one file. Yet, Prof. Mitchell speculates that "it is certainly possible that some amount of trial and error went into figuring out how to arrange the tests in this code." Unlike Prof. Mitchell, I did study this code, and as described in my Opening Report, this function is very simple and would likely not have required much trial and error for an experienced programmer like Josh Bloch to write this function.

49. Prof. Mitchell also misstates the role of APIs in Java's complexity. He says that "the use of APIs is the core structuring concept that software designers use to manage their complexity." (Opposition Report ¶ 18.) APIs are certainly a part of the software design process, but their use is not the "core structuring concept" use to manage complexity. Many other techniques, such as the design patterns (mentioned in both Prof Mitchell's Opposition Report and my Rebuttal Report), are used to manage software complexity. The first significantly complex computer systems, in fact, predate the concept of an API. For example, the IBM 360 operating system was so complex that Frederick Brooks won a Turing Award (the Association for Computing Machinery's highest award) in large part for guiding its design and implementation. However, the IBM 360 system predates the use of APIs. Similarly, Brooks' famous essays about the project, still commonly used to teach computer science students about the management of complex projects, do not mention APIs by name at all. So, while APIs are a useful tool for managing complexity, it is not correct to say that they are the core structuring concept used to

manage complexity, and this should not play into an analysis of whether APIs are protectable.

E. PROF. MITCHELL ALLEGES THAT DATA FIELDS ARE NOT FUNCTIONAL

50. Prof. Mitchell comments that I did not specifically address data fields in my Opening Report. (Opposition Report ¶ 45.) The same arguments that apply to methods and other API elements also apply to data fields that are part of the public API. Like the other API elements discussed in my Opening Report, data fields are part of the interface between third party software and the underlying libraries provided by the language. As a result, they are necessary for compatibility and interoperability, and programmers expect them to be there. Software will fail if data fields are used in one implementation and then not present in another. Similarly, like the other elements, the actual expression in a data field is a simple name, tied to the data being represented. It is necessarily brief and descriptive, and, as I understand it, unprotectable.

F. PROF. MITCHELL SUGGESTS THAT ANDROID COULD HAVE USED OTHER RULES, BUT THE RULES IN QUESTION ARE PART OF THE UNPROTECTED JAVA LANGUAGE

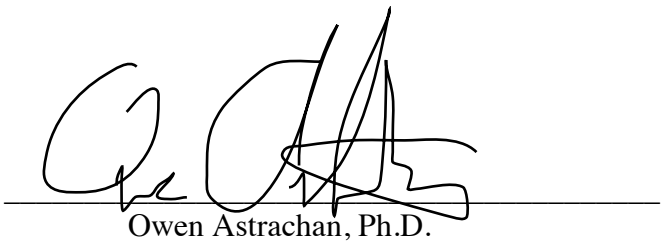
51. Prof. Mitchell responds to my discussion of the Java language's style rules by noting that Google could have "come up with its own style rules." (Opposition Report ¶ 43.) It is my understanding that Oracle has conceded that the Java language is not protected. The Java language rules are found in the book called the *Java Language Specification* and are unambiguously part of the language itself, enforced primarily by convention. Choosing a different set of rules would be tantamount to choosing a different language, which I understand Oracle does not claim is necessary.

XII. CONCLUSION

52. Prof. Mitchell's confusion of creative ideas with creative expression, and his misleading statements about the nature of creativity in API design, structure, and implementation, appear to be a distraction from the important issues that he does not address: that the alleged copying at issue is very minor portions of 12 out of 9,479 files, and methods of operation for which expression is limited by many factors, including efficiency and industry requirements.

53. I reserve the right to update and refine my opinions and analyses based on any additional materials or information that may come to my attention in the future, including additional contentions by Oracle as well as any rulings issued by the Court in this case. I also reserve the right to supplement my opinions and analyses as set forth in this report in light of any expert reports submitted by Oracle and in light of any deposition or trial testimony of Oracle's experts.

DATED: August 19, 2011



Owen Astrachan, Ph.D.